Nagy Z, Kék L, Kincses Z, Szolgay P CNN model on cell multiprocessor array. In: IEEE ECCTD 2007. European conference on circuit theory and design. Konferencia helye, ideje: Sevilla, Spanyolország, 2007 Sevilla: IEEE, 2007. pp. 276-279.

DOI: 10.1109/ECCTD.2007.4529590

CNN Model on Cell Multiprocessor Array

Zoltán Nagy, László Kék[+], Zoltán Kincses [++], Péter Szolgay[+]
Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, Hungary
+Also with Dept. Information Technology, Pázmány Péter Catholic University, Budapest, Hungary
++ Dept. Image Processing and Neurocomputing, University of Pannonia, Veszprém, Hungary

***Abstract*—Array computers can be useful in the solution of numerical spatio-temporal problems such as the state equation of the CNN or partial differential equations (PDE). IBM has recently introduced the Cell Broadband Engine (Cell BE) Architecture which contains 8 identical vector processors in an array structure. In the paper the implementation of CNN simulation kernel on the Cell BE is described. The simulation kernel is optimized, according to the special requirements of the Cell BE and can use linear and also nonlinear (piecewise linear) templates. The area/speed/power tradeoffs of our solution and different hardware implementations are also compared.**

I. Introduction

The complexity and size of a computing system increase on a chip, due to the scaling down of the geometry of the basic building blocks, the transistors. This process however, has some architectural consequences, namely, the distribution of the critical signals or limitation of dissipated power.

Array processing can be a good candidate to solve architectural problems (distribution of control signals on a chip) and to increase the computing power by using parallel computation. Different kind of architectures may be useful to organize parallel computation and execute a program. A heterogeneous array processor architecture is a good alternative for it.

II. Cell Processor Architecture

A. Cell Processor Chip

A heterogeneous multi-processor architecture was designed from one general purpose 64-bit processor called Power Processor Element (PPE) and from 8 Synergistic Processor Elements (SPEs) as shown in Figure 1. The whole architecture consists of 241M transistors, and the chip area is 235mm^2.

The operating system of PPE boosted up with Vector Multimedia Extension (VMX) technology provides a general programming/running environment, while the SPEs are SIMD[1]-only machines. High speed, 96 B/cycle, on-chip interconnection supports the effective data transfer between the processors.

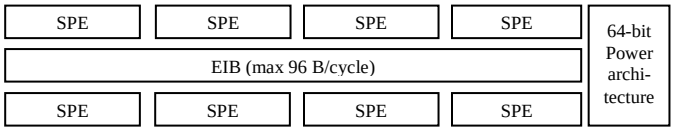

Figure 1. The heterogeneous multi-core architecture − the power processor (PPE) and 8 Synergetic Processor Elements (SPEs)

B. Cell Blade Systems

Cell blade systems are built up from two Cell processor chips interconnected with a broadband interface.

III. Problem Statement

Our primary goal is to get an efficient CNN [1] [2] implementation on the Cell Broadband Engine Architecture [3] (CBEA, or informally, CELL). Consider the CNN model and its hardware effective discretization in time.

[1] SIMD – Single Instruction Multiple Data

A. Linear templates

The state equation of the original Chua-Yang model [1] is as follows:

$$\dot{x}_{ij}(t) = -x_{ij} + \sum_{C(kl)\in N_r(i,j)} \mathbf{A}_{ij,kl} y_{kl}(t) + \sum_{C(kl)\in N_r(i,j)} \mathbf{B}_{ij,kl} u_{kl} + z_{ij} \quad (1)$$

where u_{kl}, x_{ij}, and y_{kl} are the input, the state, and the output variables. **A** and **B** are the feed-back and feed-forward templates, and z_{ij} is the bias term. $N_r(i,j)$ is the set of neighboring cells of the $(i,j)^{\text{th}}$ cell.

The discretized form of the original state equation (1) is derived by using the forward Euler form. It is as follows:

$$x_{ij}(n+1) = (1-h)x_{ij}(n) + h\left(\sum_{C(kl)\in N_r(i,j)} \mathbf{A}_{ij,kl} y_{kl}(n) + \sum_{C(kl)\in N_r(i,j)} \mathbf{B}_{ij,kl} u_{kl} + z_{ij} \right) \quad (2)$$

In order to simplify computation variables are eliminated as far as possible. First of all, the Chua-Yang model is changed to the Full Signal Range (FSR) [4] model. Here the state and the output of the CNN are equal. In cases when the state is about to go to saturation, the state variable is simply truncated. In this way the absolute value of the sate variable cannot exceed +1. The discretized version of the CNN state equation with FSR model is as follows:

$$x_{ij}(n+1) = \begin{cases} 1 & if \;\; v_{ij}(n) > 1 \\ v_{ij}(k) & if \;\; |v_{ij}(n)| \le 1 \\ -1 & if \;\; v_{ij}(n) < -1 \end{cases}$$

$$v_{ij}(n) = (1-h)x_{ij}(n) + h\left(\sum_{C(kl)\in N_r(i,j)} \mathbf{A}_{ij,kl} x_{kl}(n) + \sum_{C(kl)\in N_r(i,j)} \mathbf{B}_{ij,kl} u_{kl} + z_{ij} \right) \quad (3)$$

Now the *x* and *y* variables are combined by introducing a truncation, which is simple in the digital world from computational aspect. In addition, the *h* and (1-*h*) terms are included into the **A** and **B** template matrices resulting templates $\hat{A}, \hat{B}$.

By using these modified template matrices, the iteration scheme is simplified to a 3x3 convolution plus an extra addition:

$$v_{ij}(n+1) = \sum_{C(kl)\in N_r(i,j)} \hat{\mathbf{A}}_{ij,kl} x_{kl}(n) + g_{ij} \quad (4.1)$$

$$g_{ij} = \sum_{C(kl)\in N_r(i,j)} \hat{\mathbf{B}}_{ij,kl} u_{kl} + h z_{ij} \quad (4.2)$$

B. Nonlinear templates

Nonlinear CNN theory was invented by Roska and Chua [2]. In some interesting spatio-temporal problems (Navier-Stokes equations) the nonlinear templates (nonlinear interactions) play key role. In general. the nonlinear CNN template values are defined by an arbitrary nonlinear function of input variables (nonlinear **B** template), output variables (nonlinear **A** template) or state variables. The survey of the nonlinear templates shows that in many cases the nonlinear template values depend on the difference of the value of the currently processed cell (C_{ij}) and the value of the cell belonging to the current template element (C_{kl}). The CNN Template Library [11] contains zero- and first-order nonlinear templates.

In case of the zero-order nonlinear templates, the nonlinear functions of the template contain horizontal segments only as shown in Figure 2. This kind of nonlinearity can be used, e.g., for grayscale contour detection [11].

In case of the first-order nonlinear templates, the nonlinearity of the template contains straight line segments as shown in Figure 2. This type of nonlinearity is used, e.g., in the global maximum finder template [11]. Naturally, there exists some nonlinear templates where the template elements are defined by two or more nonlinearities, e.g., the grayscale diagonal line detector [11].

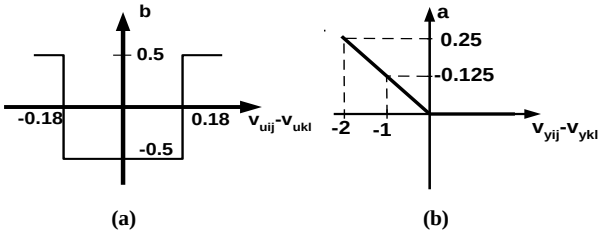

Figure 2. Zero- (a) and first-order (b) nonlinearity

IV. How to Map CNN Array to Cell Processor Array?

A. Linear Dynamics

The computation of (4.1) and (4.2) on conventional CISC processors is rather simple. The appropriate elements of the state window and the template are multiplied and the results are summed. Due to the small number of registers on these architectures, 18 Load instructions are required, which slows down the computation. Most of the CISC architectures provide SIMD extensions to speed computation up, but the usefulness of these optimizations is also limited by the small amount of registers.

The large (128-entry) register file of the SPE makes it possible to store the neighborhood of the currently processed cell and the template elements. The number of load instructions can be significantly decreased.

The SPEs in the CELL architecture are SIMD-only units, hence the state values of the cells should be grouped into vectors. The size of the registers is 128bit and 32bit floating point numbers are used during the computation, accordingly, our vectors contain 4 elements.

It seems to be obvious to pack 4 neighboring cells into one vector. However, constructing the vector which contains the left and right neighbors of the cells is somewhat complicated because 2 “rotate” and 1 “select” instructions are needed to generate the required vector (see Figure 3.).This limits the utilization of the floating-point pipeline because 3 integer instructions (rotate and select) must be carried out

before issuing a floating-point multiply-and-accumulate (MAC) instruction.

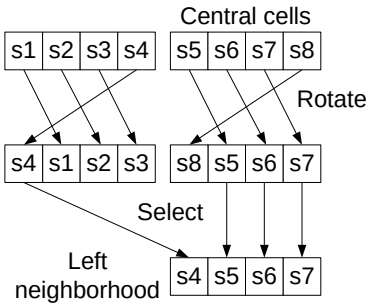

Figure 3. Generation of the left neighborhood

This limitation can be removed by slicing the CNN cell array into 4 vertical stripes and rearranging the cell values. In the above case, the 4-element vector contains data from the 4 different slices as shown in Figure 4. This makes it possible to eliminate the shift and shuffle operations to create the neighborhood of the cells in the vector. The rearrangement should be carried out only once, at the beginning of the computation and can be carried out by the PPE. Though, this solution improves the performance of the simulation data, dependency between the successive MACs still cause floating-point pipeline stalls. In order to eliminate this dependency the inner loop of the computation must be rolled out. Instead of waiting for a result of the first MAC, the computation of the next group of cells is started. The level of unrolling is limited by the size of the register file.

To measure the performance of the simulation a 256x256 sized cell array was used and 10 forward Euler iterations were computed, using a diffusion template. Without unrolling, more than 13 million clock cycles are required and the utilization of the SPE is 35%. Most of the time, the SPE is stalled, due to data dependency. By unrolling the inner loop of the computation and computing 2, 4 or 8 sets of cells, the required clock cycles can be reduced to 3.5 million and the efficiency of the SPE is nearly 90%.

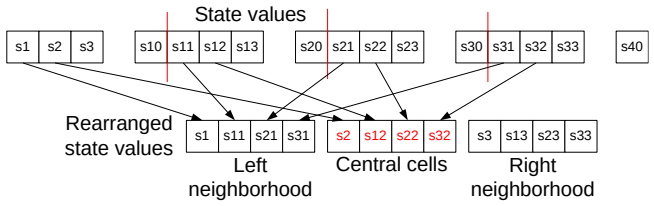

Figure 4. Rearrangement of the state values

To measure the performance of the optimized program 16 iterations were computed on a 256x256 sized cell array. The number of required clock cycles is summarized in Figure 5. By using only one SPE, the computation is carried out in 3.3 million clock cycles or 1.04ms, assuming 3.2GHz clock frequency.

To achieve even faster computation multiple SPEs can be used. The data can be partitioned between the SPEs by horizontally striping the CNN cell array. The communication of the state values is required between the adjacent SPEs when the first or last line of the stripe is computed. Due to the row-wise arrangement of the state values, this communication between the adjacent SPEs can be carried out by a single DMA operation.

By using 2 SPEs to perform the computation, the cycle count is reduced about by half, and nearly linear speedup can be achieved. However, in case of 4 or 8 SPEs the performance cannot be improved. When 4 SPEs are used, SPE number 2 requires more than 5 million clock cycles to compute its stripe. This is larger than the cycle count in case of a single SPE and the performance is degraded.

The examination of the utilization of the SPEs shows that SPE 1 and SPE 2 stall, most of the time, for the completion of the memory operations (channel stall cycle). The utilization of these SPEs is less than 15%, while the other SPEs are of efficiency similar to that of in the case of a single SPE. Investigating the required memory bandwidth shows that one SPE requires 7.2Gb/s memory I/O bandwidth and the available 25.6Gb/s bandwidth is not enough to support all 4 SPEs.

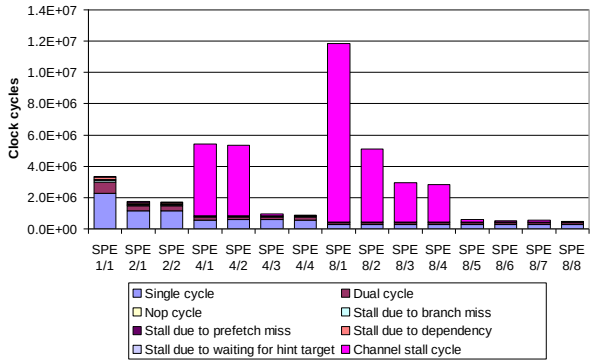

Figure 5. Intruction histogram in case of one and multiple SPEs

To reduce this high bandwidth requirement pipelining technique can be used. In this case the SPEs are chained one after the other, and each SPE computes a different iteration step, using the results of the previous SPE. Only the first and last SPE in the pipeline should access main memory. Due to the ring structure of the Element Interconnect Bus (EIB), communication between the neighboring SPEs is very efficient. The performance of the implemented CNN simulator is summarized in Figure 6.

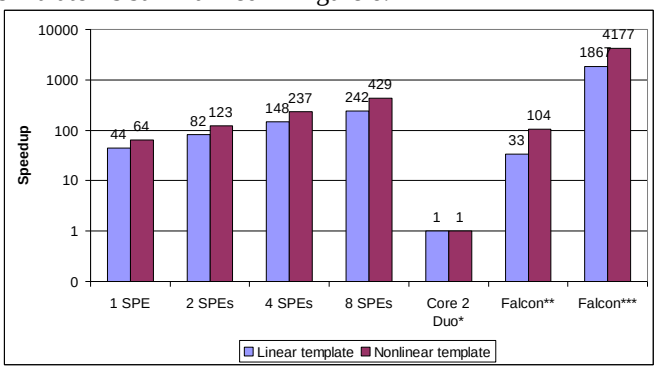

Figure 6. Performance of the implemented CNN simlator on the CELL architecture compared to other architectures (CNN cell array size: 256x256, 16 forward Euler iterations, *Core 2 Duo T7200 @2GHz, **Falcon Emulated Digital CNN-UM implemented on Xilinx Virtex-4 FPGA

(XC4VSX55) @500MHz 1 PE, ***Falcon Emulated Digital CNN-UM implemented on Xilinx Virtex-4 FPGA (XC4VSX55) @500MHz 56 PE)

B. Nonlinear Dynamics

To make using zero- and first-order nonlinear templates possible on a conventional scalar processor or on the CELL processor, the nonlinear functions belonging to the templates should be stored in Look Up Tables (LUTs).

In case of conventional scalar processors, each kind of nonlinearity should be partitioned into segments, according to the number of intervals it contains. The parameters of the nonlinear function and the boundary points should be stored in LUTs for each nonlinear template element. In case of the zero-order nonlinear templates, only one parameter should be stored in the LUT, while in the case of the first-order nonlinearity, the gradient value and the constant shift of the current section should be stored. By using this arrangement, for zero-order nonlinear templates, the difference of the value of the currently processed cell and the value of the cell belonging to the current template element should be compared to the boundary points. The result of this comparison is used to acquire the adequate nonlinear value. In case of the first-order nonlinear template, additional computation is required. After identifying the proper interval of nonlinearity, the difference should be multiplied by the gradient value and added to the constant.

Since the SPEs on the CELL processor are vector processors, the values of the nonlinear function and the boundary points are also stored as a 4-element vector. In each step four differences are computed in parallel and all boundary points must be examined to determine the four nonlinear template elements. To get an efficient implementation, optimization techniques similar to that of the linear template implementation (double buffering, vectorization, loop unrolling) can be used.

The performance of the implementation on the CELL architecture was tested by running the global maximum finder template on a 256x256 image for 16 iterations. The achievable performance of the CELL using different number of SPUs is compared to the performance of the Intel Core 2 Duo T7200 2GHz scalar processor and the nonlinear Falcon Emulated Digital CNN-UM architecture. The results are shown in Figure 6.

TABLE I. COMPARISON OF DIFFERENT CNN IMPLEMANTATIONS: 2GHZ CORE 2 DUO PROCESSOR, EMULATED DIGITAL CNN RUNNING ON CELL PROCESSORS AND ON VIRTEX FPGAS, AND Q-EYE ANALOG VLSI CHIP

Parameters	CNN Implementations			
	Core 2 Duo	***Q-Eye***	***FPGA***	***CELL***
Speed (linear template, µs)	27033.6	250	14.48	111.8
Speed (nonlinear template, µs)	84691.4	-	20.27	197.33
Power (W)	65	0.1	20	85
Area (mm^2)	143	-	~389	253

(CNN cell array size: 176x144, 16 forward Euler iterations)

V. CONCLUSION AND FUTURE WORK

Basic CNN simulation kernel was successfully implemented on the CELL architecture. Using this kernel both linear and nonlinear CNN arrays can be simulated. The kernel was optimized according to the special requirements of the CELL architecture.

The comparison of the different CNN implementation can be seen on Table 1. The comparison of the performance of the single SPE solution to a high performance microprocessor showed that about 44 times speedup can be achieved. By using all the 8 SPEs about 242 times speedup can be achieved. Compared to emulated digital architectures one SPE can outperform a single Falcon Emulated Digital CNN-UM core. When using nonlinear templates the performance advantage of the CELL architecture is much higher. In a single SPE configuration 64 times speedup can be achieved while using 8 SPEs the performance is 429 times higher.

In the future we would like to extend the capabilities of the simulator to handle large neighborhood templates. Additionally the CELL architecture is going to be used to simulate spatio-temporal dynamical problems like PDEs (for example Navier-Stokes equations).

ACKNOWLEDGMENT

The authors would like to thank Professor Tamás Roska for many helpful discussions and suggestions.

REFERENCES

[1] T. Roska and L. O. Chua, "The CNN Universal Machine: an Analogic Array Computer", *IEEE Transaction on Circuits and Systems-II*, vol. 40, pp. 163-173, 1993.

[2] T. Roska and L. O. Chua, "Cellular Neural networks with nonlinear and delay-type template elements and non-uniform girds," Int. J. Circuit Theory and Applications, vol. 20, pp. 469-481, 1992

[3] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. IBM Journal of Research and Development [Online] http://www.research.ibm.com/journal/rd/494/kahle.html, 2005.

[4] S. Espejo et.al. "A VLSI-Oriented Continuous-Time CNN Model", Int. Journal of Circuit Theory and Applications, vol. 24, 341-356, 1996.

[5] P. Szolgay, G. Vörös, and Gy. Erőss "On the Applications of the Cellular Neural Network Paradigm in Mechanical Vibrating System", *IEEE. Trans. Circuits and Systems-I, Fundamental Theory and Applications*, vol. 40, no. 3, pp. 222-227, 1993.

[6] Z. Nagy and P. Szolgay, "Numerical solution of a class of PDEs by using emulated digital CNN-UM on FPGAs", *Proc. Of 16th European Conf. On Circuits Theory and Design*, Cracow, vol. II, pp. 181-184, September 1-4, 2003.

[7] Z. Nagy and P. Szolgay, "Configurable Multi-layer CNN-UM Emulator on FPGA", *IEEE Transaction on Circuit and Systems I: Fundamental Theory and Applications*, vol. 50, pp. 774-778, 2003.

[8] Z. Nagy and P. Szolgay, "Solving Partial Differential Equations on Emulated Digital CNN-UM Architectures", *Functional Differential Equations*, vol. 13, No. 1, pp. 61-87 (2006)

[9] P. Kozma, P. Sonkoly, and P. Szolgay, "Seismic Wave Modeling on CNN-UM Architecture", *Functional Differential Equations*, vol. 13, No. 1, pp. 43-60 (2006)

[10] Z. Nagy, Zs. Vörösházi, and P. Szolgay, "Emulated Digital CNN-UM Solution of Partial Differential Equations", *Int. J. CTA*, vol. 34, No. 4, pp. 445-470 (2006)

[11] CNN Software Library (CSL) Templates (Version 1.1) [Online] Available: http://lab.analogic.sztaki.hu/Candy/csl.html